

```
EEEEEEEEEE XX XX AAAAAA MM MM PPPPPPPP LL EEEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AAAAAA MM MM PPPPPPPP LL EEEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AAAAAA MM MM PPPPPPPP LL EEEEEEEEEE SSSSSSSS
EE XX XX AA AA MMMM MMMM PP PP LL EE SS
EE XX XX AA AA MMMM MMMM PP PP LL EE SS
EE XX XX AA AA MMMM MMMM PP PP LL EE SS
EE XX XX AA AA MM MM PP PP LL EE SS
EE XX XX AA AA MM MM PP PP LL EE SS
EEEEEEEEEE XX XX AA AA MM MM PPPPPPPP LL EEEEEEEEE SSSSSS
EEEEEEEEEE XX XX AA AA MM MM PPPPPPPP LL EEEEEEEEE SSSSSS
EEEEEEEEEE XX XX AA AA MM MM PPPPPPPP LL EEEEEEEEE SSSSSS
EE XX XX AAAAAAAAAA MM MM PP LL EE SS
EE XX XX AAAAAAAAAA MM MM PP LL EE SS
EE XX XX AAAAAAAAAA MM MM PP LL EE SS
EE XX XX AA AA MM MM PP LL EE SS
EE XX XX AA AA MM MM PP LL EE SS
EEEEEEEEEE XX XX AA AA MM MM PP LLLLLLLLLL EEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AA AA MM MM PP LLLLLLLLLL EEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AA AA MM MM PP LLLLLLLLLL EEEEEEEEE SSSSSSSS
```

```

MM      MM      AAAAAA      RRRRRRRR
MM      MM      AAAAAA      RRRRRRRR
MMMM    MMMM    AA          AA      RR          RR
MMMM    MMMM    AA          AA      RR          RR
MM      MM      MM      AA          AA      RR          RR
MM      MM      MM      AA          AA      RR          RR
MM      MM      MM      AA          AA      RRRRRRRR
MM      MM      MM      AA          AA      RRRRRRRR
MM      MM      MM      AAAAAAAAAA      RR      RR
MM      MM      MM      AAAAAAAAAA      RR      RR
MM      MM      MM      AA          AA      RR          RR
MM      MM      MM      AA          AA      RR          RR
MM      MM      MM      AA          AA      RR          RR
MM      MM      MM      AA          AA      RR          RR

```


.title scrft Screen package for foreign terminals
.ident 'V04-000'
.sbttl copyright notice

* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
* ALL RIGHTS RESERVED. *

THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
TRANSFERRED.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
CORPORATION.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.

.page
.sbttl Program description

Facility

Screen Package for foreign terminal functions

Environment

Native mode, User mode, Shared library routines

When this program is linked, the image should be copied to
SYS\$LIBRARY:SCRFT.EXE and, if desired, installed to improve
the image activation speed. The screen package will automatically
"merge" this image into the calling image's address space when
needed. To debug it, simply link it /DEBUG and define a logical name
SCRFT to be the file spec of the test image. NOTE: The debugger
performs a \$EXIT system service upon exit, therefore you will only
be able to debug the first invocation of SCRFT. Also note that if
this is going to be used with any images installed with privilege,
such as SHOW PROC/CONT, PHONE, or MONITOR, SCRFT needs to be installed.

Author

Tim Halvorsen, February 1980

Modified by

V03-001 BLS0186

Benn Schreiber

21-Sep-1982

: Implement put_line and up_scroll. Allow multiple strings
: to be inserted into output buffer.
:--

.sbtll Symbol Definitions

```

;
;
;

```

Symbol Defintions

\$scrdef

; Screen package definitions

```

bs      =      ^x08      ; Backspace
lf      =      ^x0a      ; Line feed
cr      =      ^x0d      ; Carriage return
esc     =      ^x1b      ; Escape character
cb      =      ^x1f      ; Line/column bias in message
lb      =      ^a'['      ; Left bracket (vt100)
semi    =      ^x3b      ; Semi-colon (vt100)

```

```

VT100_SC = ^A'f'      ; VT100 set cursor position
VT100_DWN = ^A'M'      ; VT100 down scroll
VT100_EOS = ^A'J'      ; VT100 erase to end of screen
VT100_EOL = ^A'K'      ; VT100 erase to end of line

```

```

SAMPLE_SC      = ^A'Z'      ; Sample set cursor position
SAMPLE_HOME    = ^A'a'      ; Sample set to home and erase screen
SAMPLE_EOS     = ^A'F'      ; Sample erase to end of screen
SAMPLE_EOL     = ^A'E'      ; Sample erase to end of line

```

```

;
;
;

```

Macro definitions

```

;macro string text,?l1,?l2
;save
;psct _scr$strings,exe,nowrt,pic
l1:  .ascii text
l2:
;restore
;long l2-l1
;long l1-<.-4>      ; self-relative address
;endm string

;macro case,src,displist,type=w,limit=#0,nmode=s^#.?base.?max
case'type      src,limit,nmode'<<max-base>/2>-1
base:
;irp ep,<displist>
;word ep-base
;endr
max:
;endm
;psct _scr$code,exe,nowrt,pic

```



```
.sbtll DISPATCHER    Dispatch to proper request routine
```

```
---
This is the main entry point of the image. Dispatch
to the proper screen package routine for the foreign
terminal.
```

Inputs:

```
28(ap) = Address of argument list:
00) Number of longword arguments
04) Address of SCR argument list
08) Screen package request #
12) Foreign terminal #
16) Address of retbuf descriptor
20) Address of retlen word
24) Address of SCRFT data area (32 bytes for use by SCRFT,
                                allocated by SCRPKG)
```

Outputs:

```
Dispatch to routine with:
```

```
ap = Address of SCR argument list
r1 = Foreign terminal number (biased at zero; ft1=0)
r11 = Address of above argument list
```

--- dispatcher:

```
.word    ^m<r2,r3,r4,r5,r6,r11>

movl     28(ap),r11          ; r11 = address of SCRFT args
callg    @4(r11),b^10$      ; call with SCR argument list
ret

10$:     .word    0
clrw     @20(r11)            ; Clear output length
subl3    #1,12(r11),r1       ; r1 = Foreign terminal number - 1
cmpl     r1,#2               ; we only handle FT1 and FT2
bgequ    20$                 ; ignore others
case     8(r11),type=w,<-    ; Case on request type
put_screen,-                ; PUT_SCREEN
get_screen,-                ; GET_SCREEN
erase_page,-                ; ERASE_PAGE
erase_line,-                ; ERASE_LINE
set_cursor,-                ; SET_CURSOR
down_scroll,-              ; DOWN_SCROLL
20$,=                        ; SCREEN INFO - always handled by SCR
put_line,-                  ; PUT_LINE
20$,-                      ; MOVE_CURSOR (not implemented in SCRPKG)
set_scroll,-                ; SET_SCROLL
up_scroll -                 ; UP_SCROLL
>

20$:     movl     #1,r0
ret
```


.sbtll POS_OUTPUT Insert line,col in output message

This routine is used to insert a screen position
into a string before writing the message to the
terminal.

Calling sequence:

Branch from CALLED routine

Inputs:

R1 = Address of control string descriptor (address is self-relative)
4(ap) = Line number
8(ap) = Column number
r11 = SCRFT arguments

1 optional argument to the fao string may be placed
on the top of the stack on entry to this routine.

hextoasc:
 .ascic '!UL;!UL' ; Convert to VT100 sequences

pos_output:
 movl (sp),r0 ; Save optional argument
 case 12(r11),type=b,limit=#1,<- ; Case on terminal type
 20\$,- ; vt100 requires ; in middle
 30\$> ; sample use different bias

20\$: subl #16,sp ; Allocate FAO output buffer
 pushr #^m<r0,r1> ; Save registers
 pushab 9(sp) ; Create descriptor of buffer
 pushl #15 ;
 movab hextoasc,r1 ; Address of FAO control string
 movzbl (r1)+,r0 ;
 movq r0,-(sp) ; Push descriptor of string onto stack
 movl sp,r0 ; Address of stack arguments
 pushl 8(ap) ; Column number
 pushl 4(ap) ; Line number
 sfao_s (r0),8(r0),8(r0) ; Convert to ASCII characters
 movb 16(sp),32(sp) ; Copy length into ASCII string
 addl #24,sp ; Leave buffer on top of stack
 popr #^m<r0,r1> ; Restore registers
 brb 50\$

30\$: subb3 #1,8(ap),(sp) ; Line number
 subb3 #1,4(ap),1(sp) ; Column number
 movb #2,-(sp) ; Length of string

50\$: subl #2048,sp ; Allocate FAO buffer
 pushl sp ; Create descriptor for buffer
 pushl #2048 ;
 movq (r1),-(sp) ; Move descriptor of string on stack
 addl r1,4(sp) ; Relocate self-relative address

```

pushl    r0                ; Optional extra argument
movab    4(sp),r0
pushab   4+16+2048(sp)     ; Address of counted string
sfao_s   (r0),8(r0),8(r0)  ; Insert into output string
addl     #16,sp            ; Pop everything but fao buffer
movl     sp,r1             ; Address of descriptor
subl     sp,4(r1)          ; Make into self-relative address
brb      output

```

[illegible]


```
.sbtll OUTPUT      Write string to terminal
```

```
OUTPUT
```

```
This routine writes the specified string to the
terminal without carriage control.
```

```
Calling sequence:
```

```
Branch from CALLED routine
```

```
Inputs:
```

```
r1 = Address of string descriptor (the address is self-relative)
@20(R11) = number of bytes in output buffer already
```

```
output:
```

```
movzwl @20(r11),-(sp)      ; Get length of output written
movq    @16(r11),r2        ; Get descriptor of output buffer
subl2   (sp),r2            ; Decrease room left in buffer by used
addl2   (sp),r3            ; Update pointer to first free
cmpw    (r1),r2            ; Check if enough room in buffer
bgtru   80$               ; branch if buffer overflow
addw2   (r1),@20(r11)      ; Store length of output string
movc    (r1),@4(r1)[r1],(r3) ; store in output buffer
incl    r0                ; success
brb     90$
80$:    movl    #ss$_bufferovf,r0 ; set error
90$:    ret
```

```
.sbttl PUT_LINE      Write line to screen
```

```
PUT_LINE
```

```
This routine writes a message to the terminal
```

```
Inputs:
```

```
4(AP) = Address of descriptor of string to output
8(AP) = (OPTIONAL) Signed number of lines to advance after output
          (default is 1)
12(AP) = (OPTIONAL) Text attributes (ignored by this routine)
```

```
put_line:
    tstb    (ap)                ;Any arguments at all?
    beql    70$,r0              ; if eq! no
    movl    r1,r6               ;Save terminal type index
    pushl   #0                  ;Assume no attributes passed
    cmpb    (ap),#3             ;Were attributes passed?
    bgtru   70$,r0              ; branch if too many args passed
    blssu   10$,r0              ; branch if no attributes passed
    movl    12(ap),(sp)         ;Copy user-passed attributes
10$:    clrq    -(sp)            ;Stack column,line = 0,0
    pushl   4(ap)               ;Stack descriptor address
    calls   #4,w^call_put_screen ;Call regular routine
    blbc    r0,60$              ;Branch if error
    movl    #1,r5               ;Assume no line count
    cmpb    (ap),#1            ;Was line count supplied?
    beql    20$,r0              ;Branch if not
    movl    8(ap),r5            ;Fetch user line count
20$:    moval   w^call_up_scroll,r2 ;Assume up scroll routine
    tstl    r5                  ;Any scrolling to do?
    beql    40$,r0              ; branch if no
    bgeq    30$,r0              ;Branch if up scrolling
    mnegl   r5,r5               ;Negative scrolling--make it positive
    moval   w^call_down_scroll,r2 ; and set other routine
30$:    movl    r6,r1            ;reset terminal index
    calls   #0,(r2)             ;Call scrolling routine
    blbc    r0,60$              ;Branch if error
    sobgtr  r5,20$              ;Loop till done
40$:    movaq   b^cr_table[r6],r1 ;Get <cr> string
    brw     output              ; output and return

50$:    movl    #1,r0            ;Return success
60$:    ret
70$:
invarg:  movl    #lib$_invarg,r0 ;Return invalid argument
    ret

cr_table:
    string  <<cr>>             ;Vt100
    string  <' ' >              ;Sample
```



```
.sbttl ERASE_PAGE      Erase to end of screen
```

```
ERASE_PAGE
```

```
This routine causes the screen to be erased
from the specified position to the end of the
screen.  If the position is not specified, the
current screen position is assumed.
```

```
Inputs:
```

```
(AP)   = Number of arguments
4(AP)  = Line number (optional)
8(AP)  = Column number (optional)
```

```
If no arguments are specified, the current cursor
position is assumed.
```

```
---
erase_page:
    tstb    (ap)                ; Any arguments specified?
    bneq    10$                 ; Branch if so
    movaq   b^erase_table[r1],r1 ; Address of output string
    brw     output
10$:
    movaq   b^erase_table2[r1],r1 ; Output string with 2 args
    brw     pos_output

erase_table:
    string  <<esc><lb><vt100_eos>> ; ERASE TO END OF SCREEN
    string  <<esc><sample_eos>>    ; VT100
    string  <<esc><sample_eos>>    ; Sample
erase_table2:
    STRING  <<esc><lb>'!AC'<vt100_sc>- ; ERASE FROM POSITION
    string  <<esc><lb><vt100_eos>>    ; vt100
    string  <<esc><sample_sc>'!AC'<esc><sample_eos>>; Sample
```

```
.sbtll ERASE_LINE      Erase to end of line
```

```
ERASE_LINE
```

```
This routine causes the screen to be erased
from the specified position to the end of the
line.  If the position is not specified, the
current screen position is assumed.
```

```
Inputs:
```

```
(AP)   = Number of arguments
4(AP)  = Line number (optional)
8(AP)  = Column number (optional)
```

```
If no arguments are specified, the current cursor
position is assumed.
```

```
---
erase_line:
    tstb    (ap)           ; Any arguments specified?
    bneq    10$           ; Branch if so
    movaq   b^line_table[r1],r1 ; Address of output string
    brw     output
10$:
    movaq   b^line_table2[r1],r1 ; Output string with 2 args
    brw     pos_output

line_table:
    string  <<esc><lb><vt100_eol>> ; ERASE TO END OF LINE
    string  <'> ; vt100
    string  <'> ; Sample
line_table2:
    string  <<esc><lb>'!AC''<vt100_sc>- ; ERASE FROM POSITION
    string  <esc><lb><vt100_eol>>- ; vt100
    string  <'> ; Sample
```



```
.sbtll SET_CURSOR      Set cursor position
```

```
SET_CURSOR
```

```
This routine causes the cursor to be moved to  
to the specified position.
```

```
Inputs:
```

```
4(AP)  = Line number  
8(AP)  = Column number
```

```
set_cursor:  
    movaq    b^cursor_table[r1],r1    ; Output string with 2 args  
    brw      pos_output
```

```
cursor_table:  
    string   <<esc><lb>'!AC'<vt100_sc>> ; SET CURSOR  
    string   <<esc><sample_sc>'!AC'> ; Sample
```

.sbtll DOWN_SCROLL Move cursor up, scrolling downward

DOWN_SCROLL

This routine causes the cursor to be moved up 1 line to same column of the previous line. If the cursor was on the top line to begin with, it stays where it was, but all the information on the screen appears to move down one line. The information that was on the bottom line of the screen is lost and a blank line appears at the top.

Inputs:

None

```
call_down_scroll:      ; CALLx entry to down_scroll
    .word 0             ; Don't need to save registers

down_scroll:
    movaq b^scroll_table[r1],r1 ; Output string with 2 args
    brw   output

scroll_table:
    string <<esc><vt100_dwn>> ; DOWN SCROLL
    string <"">                ; VT100
    string <"">                ; Sample - do nothing
```


.sbtll UP_SCROLL Move cursor down, scrolling upward

UP_SCROLL

This routine causes the cursor to be moved down 1 line to the same column of the following line. If the cursor was on the bottom line to begin with, it stays where it was, but all the information on the screen appears to move up one line. The information that was on the top line of the screen is lost and a blank line appears at the bottom.

call_up_scroll: ;CALLx entry point
 .word 0

up_scroll:
 movaq b^upscroll_table[r1],r1 ;Output string
 brw output

upscroll_table:
 string <<lf>> ;vt100 - linefeed
 string <' '> ;Sample - do nothing

```
.sbtll PUT_SCREEN      Write message to terminal
```

```
PUT_SCREEN
```

```
This routine is used to write messages to the
terminal.
```

```
Inputs:
```

```
4(AP) = Address of descriptor of output string
8(AP) = (OPTIONAL) Line number
12(AP) = (OPTIONAL) Column number
16(AP) = (OPTIONAL) Text attributes
```

```
call_put_screen:
.word 0
```

```
;CALLx entry point
```

```
put_screen:
```

```
    cml      (ap),#1          ; Check if only one argument
    bleq     5$              ; Branch if no coords given
    tstl     8(ap)           ; coords zero?
    bneq     10$             ; Branch if no--Real coords
5$:    movl    4(ap),r1        ; Address of descriptor
    movq     (r1),-(sp)       ; Copy descriptor to stack
    subl     sp,4(sp)         ; and make self-relative
    movl     sp,r1           ; Address of self-relative desc.
    brw      output
10$:   pushl   4(ap)           ; Optional fao argument
    addl     #4,ap            ; Pass coords in 4(ap),8(ap)
    movaq    b^put_table[r1],r1 ; Address of fao string
    brw      pos_output
```

```
PUT_TABLE:
string  <<esc><lb>'!AC'<vt100_sc>'!AS'> ; Write with embedded cursor ; vt100
string  <<esc><sample_sc>'!AC!AS'>       ; Sample
```

 .sbtll GET_SCREEN Read input string from terminal

GET_SCREEN

This routine accepts input from the terminal into
a user specified buffer. An optional prompt string
may be given.

Inputs:

4(AP) = Address of buffer descriptor

8(AP) = Address of prompt descriptor (optional)

get_screen:

 callg (ap),g^lib\$get_input ; Read terminal input
 ret

.sbtll set_scroll Set scrolling region

SET_SCROLL

This routine sets scrolling regions, and is not implemented in this version of SCRFT. Therefore, if a terminal which supports scrolling regions is used with SCRFT, it will operate like a non-scrolling terminal, such as a VT52.

set_scroll:
movl #1,r0
ret

SCRFT.MAR;1

16-SEP-1984 17:04:28.04 Page 17

.end dispatcher

TDR1

++ TD

FL

Ir

OU

TD_I

TH
th

